CENTRE
DE RECHERCHE
SUR LES TRANSPORTS
(CRT)
CENTRE
FOR RESEARCH
ON TRANSPORTATION

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE

C.P. 6128, SUCCURSALE A
MONTRÉAL (QUÉBEC)
CANADA
H3C 3J7

TÉLÉPHONE : (514) 343-7575
TÉLEX : 05-24146 BIBPOLYTEC
TÉLÉCOPIEUR (FAX) : (514) 343-7121

CRT - 653

SOLUTION APPROACHES FOR MULTIMODE MULTIPRODUCT ASSIGNMENT PROBLEMS

by

Omar Drissi-Kaïtouni

This research was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada.

Université de Montréal
Centre de recherche sur les transports - Publication #653
October 1989

90-01402/

Dépôt légal - 4e trimestre 1989

Abstract

This paper deals with the solution of multimode multiproduct assignment models; these are used to analyse freight transportation networks at regional and national levels. We evaluate the efficiency of some variants of the linear approximation algorithm, where at each iteration and for each product, one, two or three extremal points are generated and a 1, 2 or 3-dimensional search (exact or approximate) is performed. The variant where a 2-dimensional approximate search (two Partan iterations in a 2-dimensional space) appears to be the most efficient.

Finally, this paper points out some costly computations that may be avoided, in order to reduce the total execution times.

Keywords: Multimode, Multiproduct, Network, Assignment, Traffic, Equilibrium, Linear Approximation, Simplicial Decomposition, Partan.

Résumé

Cet article concerne la résolution de problèmes d'affectation multimode multiproduit qui permettent d'analyser des réseaux de transport de marchandise à un niveau national ou régional. Nous évaluons l'efficacité de plusieures variantes de l'algorithme d'approximations linéaires, où à chaque itération et pour chaque produit, un, deux ou trois points extrémaux sont générés, et une minimisation en une, deux ou trois dimensions est effectuée. La variante où l'on effectue une recherche bidirectionnelle approximative (deux itérations partan dans un espace à deux dimensions) semble la plus efficace.

Finalement, nous identifions des calculs coûteux et non nécessaires, ce qui permet de réduire les temps de calcul indépendemment de la variante utilisée.

Mots clés: multimode, multiproduit, réseau, affectation, trafic, équilibre, approximation linéaire, décomposition simpliciale, partan.

1. Introduction

In the context of the strategic planning of freight flows, Guélat, Florian and Crainic (1987) developed a multimode-multiproduct network assignment model. These authors used a decomposition by product - linear approximation algorithm to solve the problem. On the one hand, this algorithm has a poor rate of convergence in the neigbourhood of the optimal solution and hence, a large number of iterations must be performed in order to obtain an accurate solution; on the other hand, the proposed model is aimed at the analysis of transportation networks at national and regional levels, which are usually of medium to large scale, hence requiring significant computation time to perform each iteration of the algorithm. This model has been integrated in the interactive-graphic system STAN (Florian et al., 1987) for strategic planning of freight transportation at national and regional levels. Thus, it is of great interest to find ways to reduce these long computation times.

The goal of this paper is to investigate some ways of reducing the computation times of the multimode-multiproduct assignment module of STAN, either by optimizing the existing code, or by implementing variants of the linear approximation algorithm.

This paper is organized as follows: in section 2, we describe the multimode-multiproduct assignment model and the algorithm of Guélat et al. (1987). In section 3, we describe the test networks that are used for numerical experimentation. In section 4, improvements in the code that reduce computation times are proposed. In section 5, we investigate some variants of the Gauss Seidel-Linear Approximation (GSLA) algorithm to solve the problem. We conclude the paper in section 6.

We would like to note that the methods and algorithms that are proposed in this paper, in order to improve the efficiency of the assignment algorithm, are sufficiently general to be applied to other convex cost nonlinear minimization problems on large scale networks.

In the remainder of this paper, we refer to the multimode-multiproduct assignment problem as the MMMPA or the M^3PA problem.

2. The M^{3}PA problem and solution algorithm GSLA

The M^{3}PA model that has been proposed by Guélat et al. (1987) is a convex cost nonlinear minimization problem subject to flow conservation and nonnegativity constraints:

$$\min \sum_{p} s^p(v)^T v^p \tag{1}$$

$$s.t. \sum_{k \in K_{od}^{m(p)}} h_k = g_{od}^{m(p)},\ o \in O,\ d \in D,\ m(p) \in M(p),\ p \in P \tag{2}$$

$$h_k \geq 0,\ k \in K_{od}^{m(p)},\ o \in O,\ d \in D,\ m(p) \in M(p),\ p \in P \tag{3}$$

where:

O	is the set of origin nodes,
D	is the set of destination nodes,
P	is the set of products,
$M(p)$	is the set allowed modes for product p,
$m(p)$	is a subset of $M(p)$,
$K_{od}^{m(p)}$	is the set of paths from o to d by the modes in $m(p)$,
v^p	is the vector of arcs and transfers flow for product p,
h_k	is the flow on path k,
s^p	is the vector of costs on the arcs and transfers for products $p \in P$.

The notation used is the same as that of Guélat et al. (1987).

The algorithm used by Guélat et al. (1987) to solve this problem is a decomposition by block (where each block corresponds to a product) - linear approximation, and may be stated as follows:

GSLA Algorithm

Step 0 : Initialise v (initial feasible solution)

Step 1 : Major Cycle (GS):

For each $p \in P$ do

Minor cycle (LA):

- Compute the optimal extremal point w^p of the linear approximation for product p.
- Compute the linear approximation descent direction $d^p = w^p - v^p$.
- Find the optimal step $\lambda_p \in [0,1]$ along d^p, and update the flow for product p.

Step 2 : If some stopping criterion is satisfied then STOP;

Otherwise return to Step 1.

In the implementation of the algorithm, two stopping criteria are proposed (Guélat et al., 1987): (i) the number of iterations, and (ii) the value of the relative gap.

3. The Test Networks

In order to test the efficiency of the different approaches proposed in this paper, two test networks were used: one of the medium size (TN1) and one of the large size (TN2).

Test network #1 (TN1): This network is a model of the region of the Rio San Francisco (Brazil) that has the following dimensions:

- 76 centroids,
- 183 regular nodes,
- 866 links,
- 1684 transfers.

The generalized cost function that was used has the form:

$$f = 0.5 * (\text{operating cost}) + 0.5 * (\text{delay}).$$

The classes of operating cost and delay functions are given in Appendix 1.

Eight transportation modes are present on the network and may be used by each of the six (classes of) products.

An origin-destination matrix is associated with each product; these matrices are relatively sparse: 28, 14, 25, 25, 41 and 72 O/D pairs have nonzero demand for the corresponding products. Figure 1 illustrates this network.

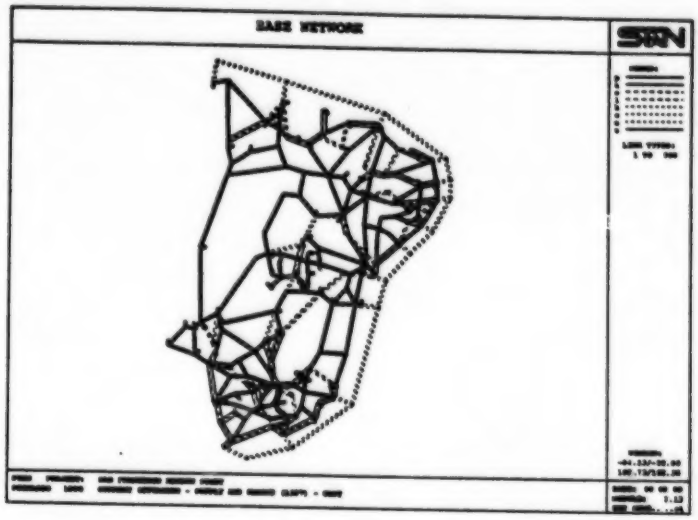


Figure 1

Test network #2 (TN2): This network is a model of the South-East region of Brazil that has the following dimensions:

- 271 centroids,
- 1243 regular nodes,
- 4957 links,
- 5718 transfers.

The generalized cost function that was used has the form: f = 1 * (delay); the classes of delay functions are given in Appendix 2.

Ten transportation modes are present on the network and may be used by each of the six (classes of) products.

An O/D demand matrix is associated with each product; these matrices are relatively

sparse: 360, 43, 368, 137, 211 and 2139 O/D pairs have nonzero demand for the corresponding products.

Figure 2 illustrates this network, and Figure 3 is a "window" on the denser network of the South-East region of TN2.

Figure 2

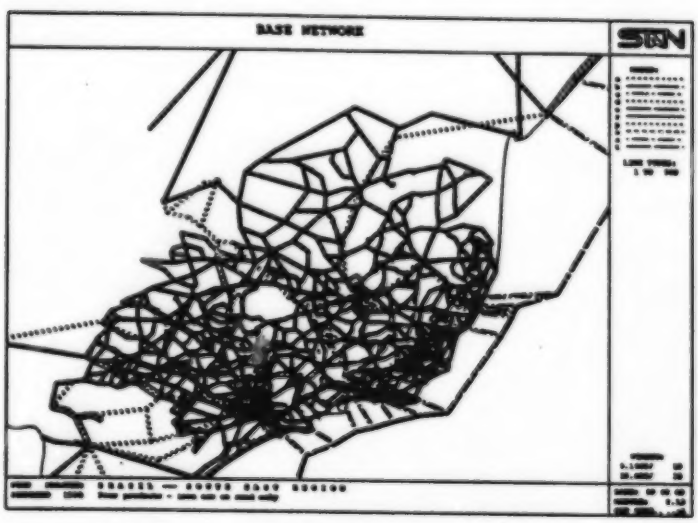


Figure 3

4. Optimization of the Existing Code

In this section, we examine the implementation of the algorithm of Guélat et al. (1987) and consider only code improvements that may reduce computation times. As a preliminary indication of the distribution of computation times in the code of the assignment algorithm of Guélat et al. (1987), Table 4.1 gives the summary for 10 iterations on network TN1; CPU times are in seconds on the SUN 3/50 (MC68020/15 Mhz) of the Centre de recherche sur les transports, Université de Montréal.

These results indicate that approximately half of the time is used to perform the line search computations, a quarter of the time to evaluate the gap, and a quarter of the time to generate extremal points (computation of marginal costs, of shortest paths and loading the demand).

Task	CPU time
Computation of marginal costs	177.0
Loading	38.2
Line search	471.4
Computation of the objective function	66.8
Computation of the gap	228.0
Total	982.3

Table 4.1

We note that the evaluation of the gap is roughly equivalent to the computation of the marginal costs and a "load" operation.

The most time consuming operation in the evaluation of the gap, in the computation of extremal points and in the line search is the evaluation of the marginal costs. Hence, it is this calculation that needs to be analysed in detail. In the following, we show that the evaluation of some marginal costs may be avoided, since not all the link marginal costs are necessary for the computation of the gap, the extremal points and carrying out the line search.

4.1 Line search

In the code of the assignment algorithm of Guélat et al. (1987), which was implemented in a first version of STAN (Florian et al., 1987), given a feasible solution v^p and an extremal point w^p, the step λ^p along the direction $d^p = w^p - v^p$ used to decrease the objective function is computed as follows: the directional derivative $D(\lambda) = \nabla f(v^p + \lambda d^p)^T d^p$ is evaluated at six points $((\lambda_i, D(\lambda_i)), \lambda_i \in [0,1], i=1,...,6)$. When $D(0)$ is positive λ^p is set to 0, and when $D(1)$ is negative λ^p is set to 1. When $D(0) < 0$ and $D(1) > 0$, the optimal step (for which $D(\lambda^p) = 0$) is between 0 and 1. Then, a quadratic approximation function is fitted through three points $(\lambda_i, D(\lambda_i))$, $(\lambda_{i+1}, D(\lambda_{i+1}))$, $(\lambda_{i+2}, D(\lambda_{i+2}))$ that satisfy $D(\lambda_i) \leq 0$ and $D(\lambda_{i+2}) \geq 0$. λ^p is then computed to be the value for which the quadratic approximation function takes the value zero.

The computation time needed to fit the quadratic function is very small, when compared to the time to compute the marginal costs $\nabla f(v^p + \lambda_i d^p)$ for each λ_i, for the evaluation of $D(\lambda_i)$, i=1,2,... . Here, we must note that only the components $\nabla f(v^p + \lambda_i d^p)_l$ and $\nabla f(v^p + \lambda_i d^p)_t$ that correspond to links l and transfers t with a $d_l^p \neq 0$ and $d_t^p \neq 0$ are needed. Thus, one needs to compute the marginal costs only for the links l with $d_l^p \neq 0$ and transfers t with $d_t^p \neq 0$. This idea seems trivial, and its impact is significant for large scale networks. Table 4.2 shows the resulting decrease in the computation times, if the above refinement is implemented.

	CPU time for the line search (10 iterations)	
	Original code (Marginal costs computed for all links and transfers)	Modified code (Marginal costs computed only for links l and transfers t such that $d_l^p \neq 0$ and $d_t^p \neq 0$)
TN1 with 2 products	471	111
TN2 with 6 products	20131	11103

Table 4.2

4.2 Computation of the Objective Function

Altough the computation times used to evaluate the objective function are not high compared to the other steps of the algorithm, it is important to reduce it as much as possible. Hence, we suggest to apply the same idea as that used for the line search; the objective function has the form $s(v)^T v$. Therefore, we suggest to compute $s_l^p(v)$ and $s_t^p(v)$ only for the links l and transfers t such that $v_l^p \neq 0$ and $v_t^p \neq 0$. The results relative to this improvement are depicted in Table 4.3.

Again, this kind of simple improvement has a significant impact in reducing the computation times.

	Times for the computation of the objective function (10 iterations)	
	Original code (s(v) computed for all the links and transfers)	Modified code ($s_l^p(v)$ and $s_t^p(v)$ computed only if $v_l^p \neq 0$ and $v_t^p \neq 0$)
TN1 with 2 products	66.8	12.4
TN2 with 6 products	468	182

Table 4.3

4.3 Marginal cost and loading

The algorithm used by Guélat et al. (1987) for the computation of shortest paths is an adaptation of Dikjstra's label setting method for a multi-modal network where transfers are allowed between modes (Guélat et al., 1987). In the implementation of the assignment algorithm, before any call to the shortest paths subroutine, the marginal costs are computed for all the links and transfers. We suggest the two following modifications in order to improve the efficiency of the code:

4.3.1 Stopping Criterion:

In the shortest path algorithm (Guélat et al., 1987) the stopping criterion is: $\bar{A} = \emptyset$, where $\bar{A}$ is the set of arcs to be labeled. We suggest to use the following criterion: ($\bar{A} = \emptyset$ or $u_{\bar{a}} \geq u_j, \forall j \in D$) where $\bar{a}$ is the arc to be labelled at the current iteration, and $u_{\bar{a}}$ is the cost of the shortest path from the origin to $\bar{a}$. D is the set of destination nodes, and u_j is the cost of the shortest path from the origin to destination node j at the current iteration of the shortest path algorithm. Since $u_{\bar{a}} = \min\{u_a, a \in \bar{A}\}$ and the marginal costs on all arcs and transfers are nonnegative, if $u_{\bar{a}} \geq u_j, j \in D$, then it is no longer possible to reduce any of the path costs $u_j, j \in D$; this means that the shortest paths from the origin to all the destinations $j \in D$ have been found.

4.3.2 Arc marginal costs computations to be avoided

By using the stopping criterion suggested above, when $u_{\bar{a}} \geq u_j, j \in D$ is satisfied, then the arcs $a \in \bar{A}$, $a \neq \bar{a}$, will not necessarily be labelled when the algorithm terminates. Thus, the marginal costs of the nonlabelled arcs need not be computed either. Unfortunately, these arcs are not known "a priori". We suggest, instead of computing all the arc marginal costs before the start of the shortest path algorithm, to compute arc marginal costs as required, during the labelling process used in the shortest path algorithm. However, one call to a subroutine that computes the marginal cost of N arcs is less time consuming than N calls to a subroutine that computes the marginal cost of one arc. Thus it may be advantageous to compute (before the start of the shortest path algorithm), in groups, the marginal costs for all the arcs that are known "a priori" to be needed in the computations. Some of these arcs may be identified as follow: as soon as a shortest path computation has been performed, one needs to compute a directional derivative $\nabla f^p(v)^T(v^p - w^p)$ for product p. Hence marginal costs must be known for all arcs a such that $v_a^p - w_a^p \neq 0$. Table 4.4 gives the values of $v_a^p - w_a^p$ given v_a^p and w_a^p:

$v_a^p - w_a^p$	$v_a^p = 0$	$v_a^p \neq 0$
$w_a^p = 0$	0	v_a^p
$w_a^p \neq 0$	$-w_a^p$	$v_a^p - w_a^p$

Table 4.4

When $w_a^p \neq 0$, then arc a belongs to a shortest path, and its marginal cost is known at the end of the shortest path algorithm. When $w_a^p = 0$, then arc a does not belong to a shortest path and the corresponding marginal cost is not necessarily known, so it must be computed if $v_a^p \neq 0$. As a consequence, the arcs a such that $v_a^p \neq 0$ are known, "a priori", to be needed further, so their marginal costs can be computed before the start of the shortest path algorithm, and it will not be necessary to recompute these individually during the shortest path computations.

The resulting decrease for 10 iterations in the computation times due to these suggestions are reported in Table 4.5.

	Times for the computing marginal costs, loading and the gap (≈ marginal costs + loading)	
	Original code	Modified code
TN1 with 2 products	443.4	376
TN2 with 6 products	27100	20169

Table 4.5

Table 4.6 summarizes the decrease of the total computation time by including all the suggestions described above, for different test networks.

Network	Number of iterations	Original code (CPU time)	Modified code (CPU time)	Reduction of total CPU time
TN1 with 2 products	10	980	500	49%
TN1 with 3 products	8	1700	900	47%
TN1 with 5 products	3	2400	1100	54%
TN2 with 6 products	10	47700	31500	34%

Table 4.6

5. Solution algorithms

The algorithm proposed by Guélat et al. (1987) to solve the M^{3}PA problem has been described in section 2, is a Gauss Seidel - Linear Approximation (GSLA) approach. An important advantage of this algorithm is that the flow conservation constraints are always satisfied. At any iteration, the (feasible) solution v^p, $p \in P$ is implicitly expressed as a linear combination of extremal points that correspond to "all or nothing assignments" on sets of shortest paths. When, an extremal point w^p is generated for product p, a step λ^p along the direction $d^p = w^p - v^p$ means that some of the flow on the arcs and transfers that correspond to v^p is transfered to the arcs and transfers that correspond to w^p. Feasibility is maintained by taking a step $\lambda^p \in [0,1]$.

Hence, this type of algorithm is easy to implement and requires relatively little memory storage, when compared to the algorithms that necessitate to handle explicitly the constraints of flow conservation (2); those require to know "a priori" all the paths, which is not realistic for large networks (e.g. 8000 links, 8000 transfers, 4000 nodes, etc., in STAN).

Therefore, in the remainder of this paper, we consider only variants of the GSLA algo-

rithm which may be characterized by the following mapping.

$$\text{GSLA} = A = A^{p_n p} \circ \ldots \circ A^{p_1}$$

where $A^p = M \circ D^p$.

D^p represents the search for a descent direction for product $p \in P$, and

M is a 1-dimensional minimization along the direction found by D^p.

In the following, we consider some variants of the algorithm GSLA that are all of a same class of algorithms that we denote GSLA(r) (r=1,2,...).

Before stating GSLA(r), r=1,2,..., we require the following notation:

- $f(v)$: the differentiable convex objective function (1) to be minimized,
- Ω : the feasible set of solutions defined by the constraints (2) and (3),
- $Lf_y(v)$: the linear approximation of $f(v)$, evaluated at point y,
- $H(v^1, ..., v^j)$: the convex hull of v^1,...,v^j.

GSLA(r), r=1,2,...:

Step 0 : Initialization: $v^0 \in \Omega, k = 0$

Step 1 : Major Cycle (GS):

For each $p \in P$ do

Minor cycle (LA(r)):

a) $z^0 = v_p^k$, $W = \{z^0\}$, $y^0 = v_p^k$

b) For $i = 1$ to r do

- Compute $z^i = argmin\ Lf_{y^{i-1}}(z)$ $s.t.$ $z \in \Omega$
- $W = W \cup \{z^i\}$
- $y^i = argmin\ f(y)$ $s.t.$ $y \in H(y^{i-1}, z^i)$

c) $v_p^{k+1} = argmin\ f(y)$ $s.t.$ $y \in H(z^0, z^1, ..., z^r)$

Step 2 : Stop, or $k = k + 1$ and goto Step 1.

This class of algorithms is in the spirit of the Restricted Simplicial Decomposition (RSD) (see Hearn and Lawphongpanich, 1984); the difference is that at each minor cycle, the set W of extremal points is reinitialized to the empty set and r Linear Approximation iterations are performed for the problem of product p, followed by a minimization over the convex hull of the extremal points generated by the r LA iterations. In the classical RSD, the extremal points z^i (i=0,...,r) are kept from an iteration to another. In the context of the M^{3}PA problems, this means that r extremal points must be kept for each of the products $p \in P$; then eventually $r \times p$ vectors (each of dimension 1 × 16000 in STAN) would be kept. In the algorithm GSLA(r), only r of those vectors are necessary at each minor cycle. Hence, the memory storage requirements are significantly reduced, compared to the RSD algorithm.

When r is fixed to 1, only one extremal point is generated so, the minimization at step 1-c is done over $H(z^0 = v^k, z^1)$ which is the classical linear approximation minimization. Thus GSLA(1)=GSLA.

Numerical results with GSLA(r) for $r > 1$ will be presented later in this section. Now, we concentrate on the convergence of GSLA(r).

Convergence proof of GSLA(r), r=1,2,...

GSLA(r) may be stated in a compact form as:

$$\text{GSLA}(r) \;=\; A_r^{p_n p} \circ \ldots \circ A_r^{p_1}$$

where $A_r^p \;=\; M_r \circ D_r^p$.

D_r^p is the computation of the space $H(z^0, z^1, \ldots, z^r)$ (in the step 1-a and 1-b), for product p, and,

M_r is a minimization of f over $H(z^0, \ldots, z^r)$ (in step 1-c).

We note that D_r^p is a sequence of r linear approximation iterations for product p, so:

$$D_r^p \;=\; \underbrace{A^p \circ A^p \circ \ldots \circ A^p}_{r \text{ times}}.$$

It is proven in Guélat et al. (1987) that A^p is closed and that every direction generated by A^p is a feasible descent direction.

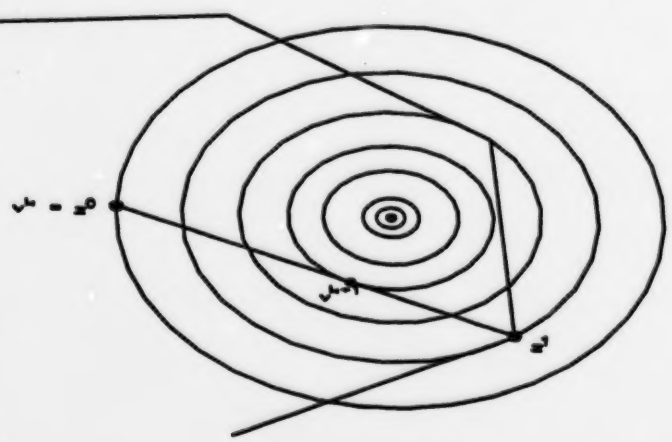
By Lemma 4.2 of Zangwill (1967), D_r^p is also closed, and thus $H(z^0, ..., z^r)$ is a feasible descent space for f (since it contains $H(z^0, z^1)$, the descent direction generated by A^p).

As for M_r, it may be any closed mapping for minimizing f over $H(z^0, ..., z^r)$ (M_r may be a Linear Approximation, Partan, Gradient, ..., algorithm over $H(z^0, ..., z^r)$). Again, by Lemma 4.2 of Zangwill (1967) on $M_r \circ D_r^p$, A^p is closed on $A = A_r^{p_m p} \circ ... \circ A_r^{p_1}$, A is closed.

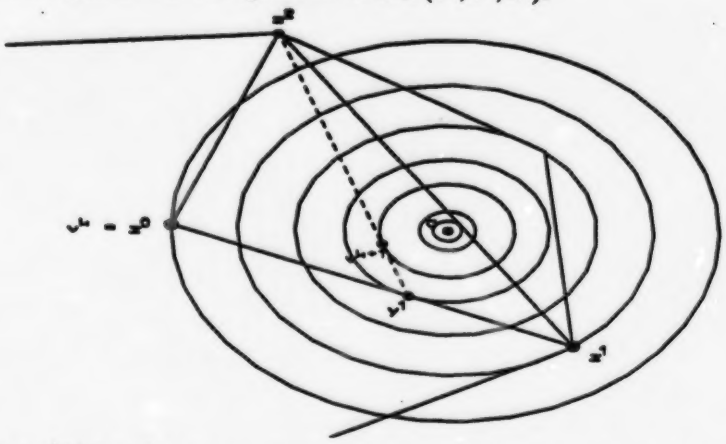
The global convergence of algorithm GSLA(r) follows from Convergence Theorem A of Zangwill (1967) and the results above.

We must remark that global convergence of this algorithm has been based on the fact that M_r is an exact minimization over $H(z^0, ..., z^r)$. This may be very time consuming for large networks. In what follows, we consider variants of this algorithm where M_r is replaced by more or less approximate minimization mappings.

Algorithm A1: r=1 and M_1 is an approximate 1-dimensional minimization over $H(z^0, z^1)$, described in detail in Guélat et al. (1987). A1 is then the algorithm GSLA proposed by Guélat et al. (1987) for the solution of the M^{3}PA problem.



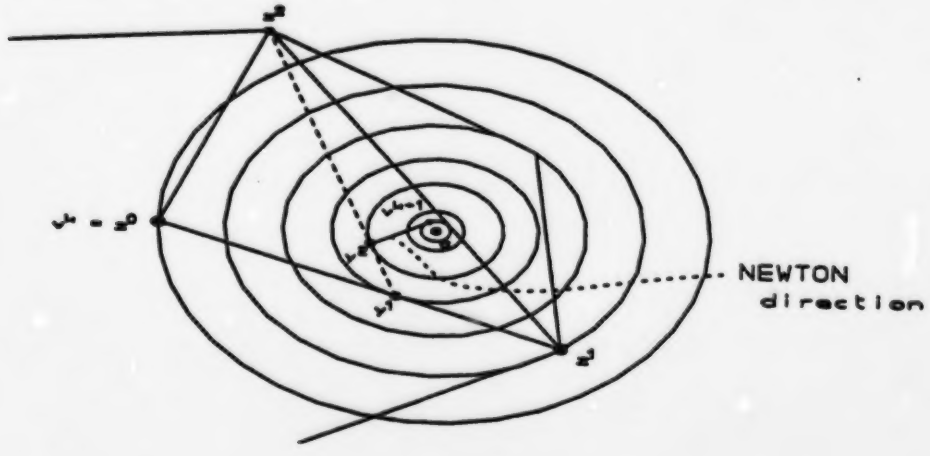
Algorithm A2: r=2 and M_2 is an approximate minimization defined by two iterations of the linear approximation algorithm over $H(z^0, z^1, z^2)$.



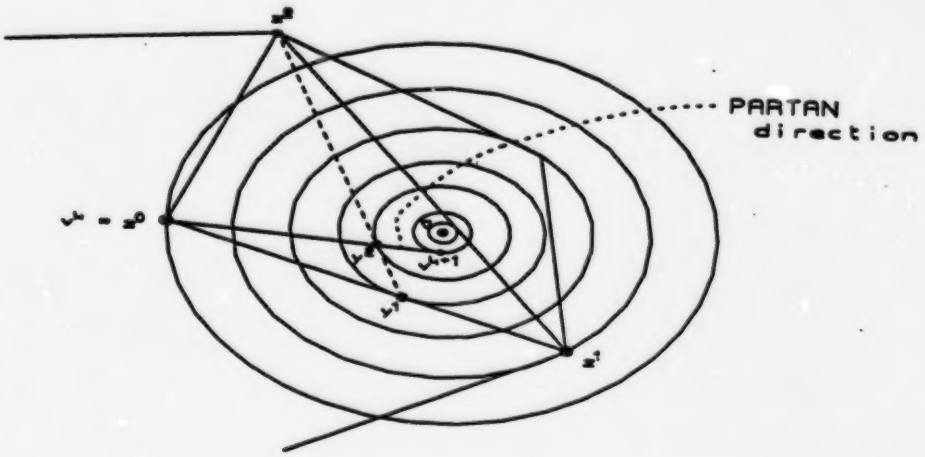
Note that v^{k+1} is only an approximate solution of the optimal solution $\bar{v}$ (on $H(z^0, z^1, z^2)$).

A2 ≡ two successive linear approximation iterations for each product p, and A2 = $(A^{p_n p} \circ A^{p_n p}) \circ \ldots \circ (A^{p_1} \circ A^{p_1})$.

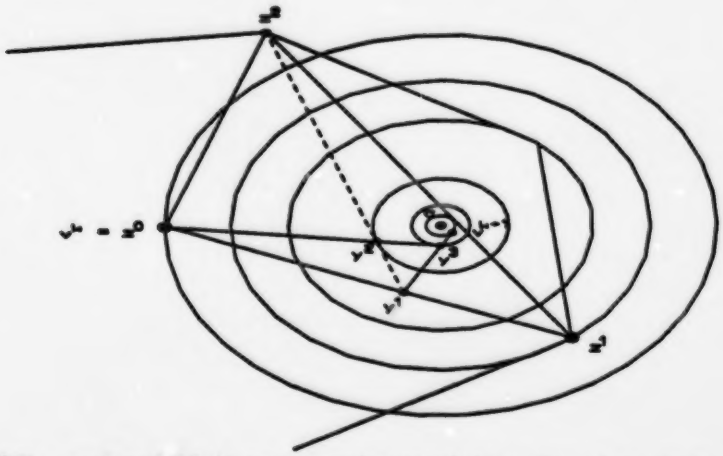
Algorithm A3: r=2 and M_2 is an approximate minimization over $H(z^0, z^1, z^2)$ defined by two linear approximation iterations plus one Newton iteration over $H(z^0, z^1, z^2)$.



Algorithm A4: r=2 and M_2 is an approximate minimization characterized by a Partan (see for instance Florian et al., 1987a) iteration over $H(z^0, z^1, z^2)$.



Algorithm A5: r=2 and M_2 is an approximate minimization characterized by two Partan steps over $H(z^0, z^1, z^2)$.



Algorithm A6: r=3 and M_3 is an approximate minimization characterized by two Partan steps over $H(z^0, z^1, z^2, z^3)$.

Any other algorithm with r=1,2,3,4,... and M_r exact or approximate minimization over $H(z^0, ..., z^r)$ may be used. However, we will restrict ourselves to the study of the six proposed algorithms A1 to A6.

6. Numerical Results

The algorithms of section 5 have been coded in Fortran 77 on the SUN 3/50 (MC68020/ 15 Mhz) of the Centre de Recherche sur les Transports, Université de Montréal. Their efficiency has been tested on the test networks TN1 and TN2 that were described in Section 2.

For TN1 we tested 3 problems with respectively 2, 3 and 5 products on the network, where each product can use all of the 8 present modes.

For the larger network TN2, we tested the largest possible problem size with 6 products on the network, where each product can use all of the 10 modes present on the network.

In the following, we present, for each test problem, the numerical results by mean of a figure (fig. 4 to 7) representing the gap vs. the computation time of the algorithms A1 to A6 (where the code refinements described in section 4 have been included). The legend used in figure 4 is the same for figures 5, 6 and 7.

The (relative) gap is computed as follows:

$$GAP = \frac{OBJ - BLB}{OBJ} \times 100$$

where OBJ is the value of the objective function at the current iteration, and BLB is an approximation of the optimal value of f, defined as the best lower bound of f found by the algorithms A1 to A6.

It appears from figures 4 to 7 that the algorithms A4 and A5 (1 or 2 Partan steps in $H(z^0, z^1, z^2)$) are the most efficient.

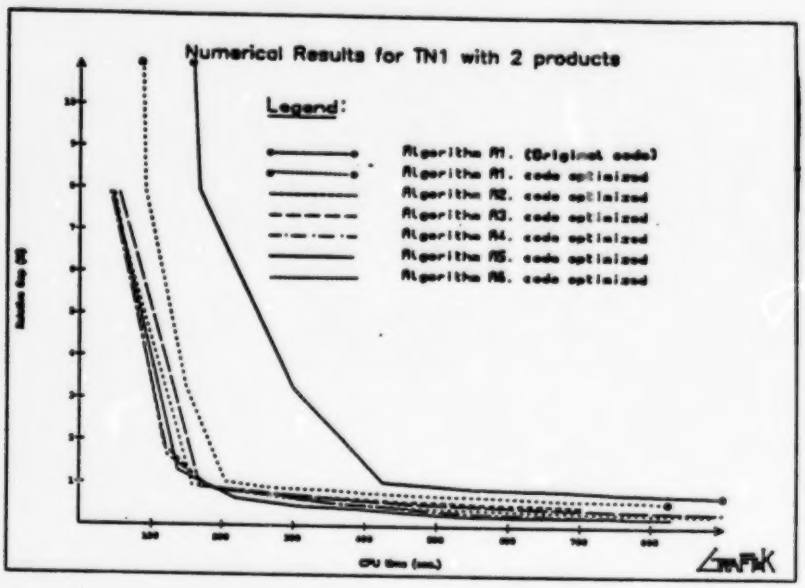


Figure 4

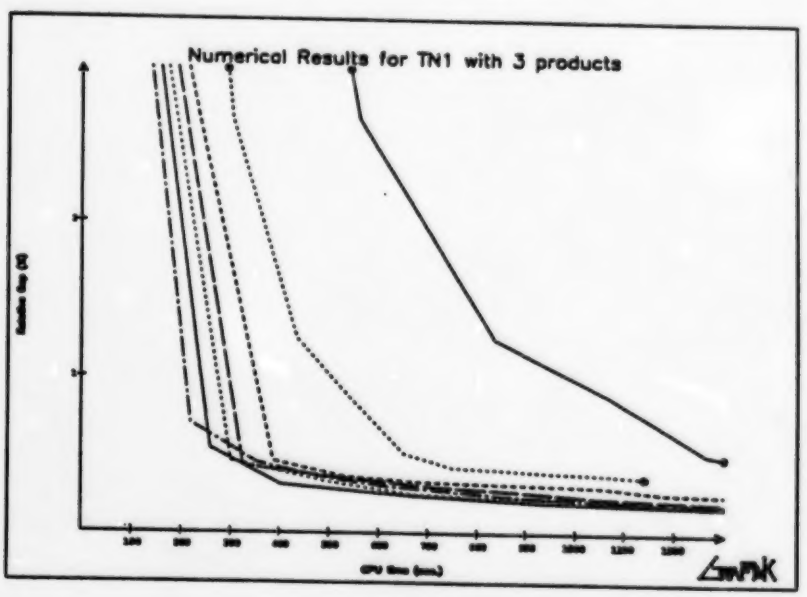


Figure 5

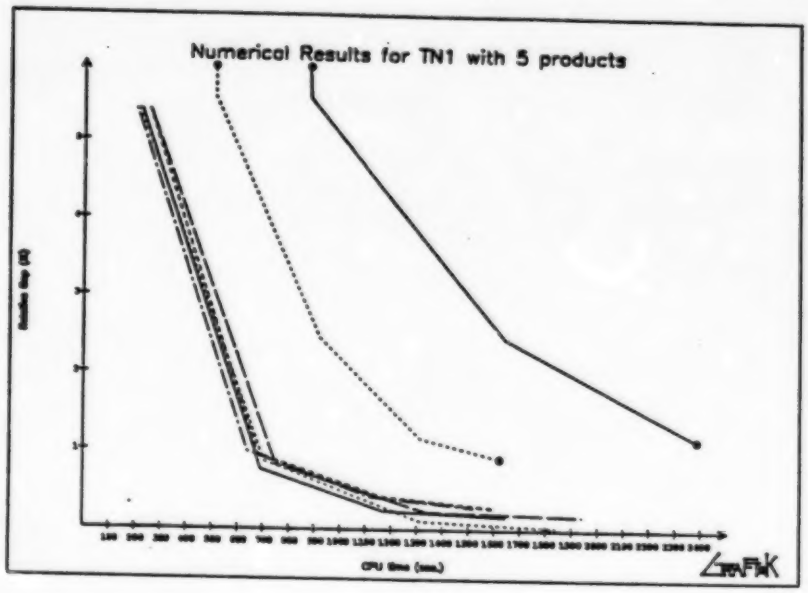


Figure 6

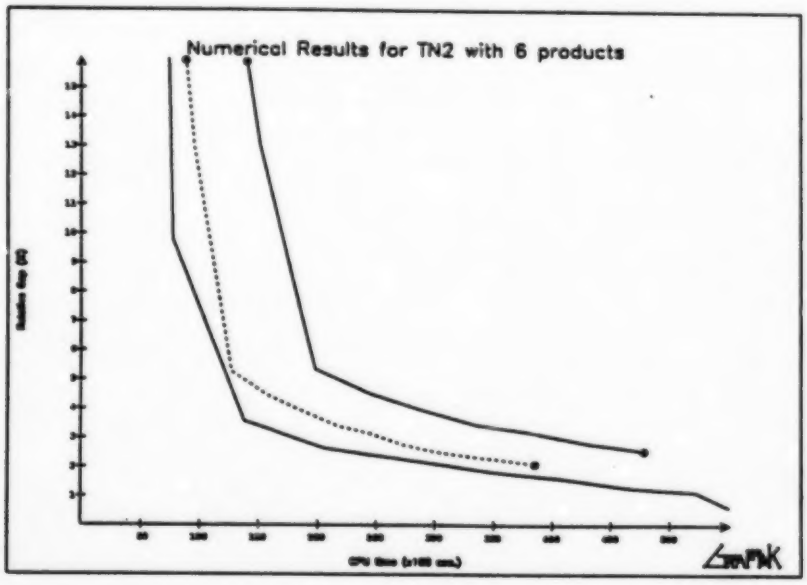


Figure 7

7. Conclusion

This paper deals with the solution of the multimode-multiproduct assignment model which is aimed at the analysis of transportation networks at national and regional levels (that are usually of medium to large scale). Guélat et al. (1987) proposed and implemented in STAN (Florian et al., 1987) a Gauss Seidel-Linear Approximation (GSLA) algorithm to solve the problem.

The aim of this paper is:

1. to present some methods to improve the implementation of the proposed GSLA algorithm (and its variants) by identifying some unnecessary time consuming computations, and
2. to evaluate the efficiency of the algorithm GSLA compared to other decomposition algorithms such as the Decomposition by product-Partan algorithms.

As for point (1), because of the size of the networks, it appears that the code may be improved significantly by simple refinements such as not computing link and transfer costs and marginal costs when they are to be multiplied by 0.

For point (2), some algorithms, in the spirit of the restricted simplicial decomposition (Hearn, 1984) are tested; at each iteration, for each product, one, two or three extremal points are generated and a minimization is performed (approximately) on the convex hull of the extremal points and the current solution. The variant that performed the best (in term of computation time) is the one where two extremal points are generated, and the optimal solution, over the convex hull of these two extremal points and the current solution, is approximated by the solution obtained after two Partan iterations in the convex hull.

Numerical results show that the improvements of points (1) and (2) help to reduce the high computation times met in M^3PA problems in particular and for large scale network optimization problems in general.

8. Bibliography

Florian, M., Crainic, T.G., Guélat, J. and Babin, A. (1987), *STAN: An Interactive-Graphic System for Strategic Transportation Analysis at the National and Regional Levels*, Publication #495, Centre de recherche sur les transports, Université de Montréal.

Florian, M., Guélat, J. and Speiss, H. (1987a), *An Efficient Implementation of the "PARTAN" Variant of the Linear Approximation Method for the Network Equilibrium Problem*, Networks17: 319-339.

Guélat, J., Florian, M. and Crainic, T.G. (1987), *A Multimode Multiproduct Network Assignment Model for Strategic Planning of Freight Flow*, Publication #549, Centre de recherche sur les transports, Université de Montréal.

Hearn, D.W., Lawphongpanich, S. and Ventura, J.A. (1984), *Restricted Simplicial Decomposition: Computation and Extensions*, Research Report no. 84-38, Industrial and Systems Engineering Department, University of Florida.

Zangwill, W.I. (1969), Nonlinear Programming: A Unified Approach, Prentice-Hall International Series in Management.

Acknowledgment

Many thanks are due to Michael Florian who supervised this project and brought valuable comments from the begining of the research to the last sentence of this paper.

Also, I am grateful to Ilana Berger who adapted the code for the algorithms A4 and A6.

Appendix 1

FUNCTIONS ON LINKS

Function_Types:	**Codes:**
Operating cost	FCLnn
Delay	FDLnn
Energy consumption	FFLnn

Variables:	
LENGTH	Link length
CAP	Link capacity
BETA	Link special coefficient
UL1,UL2,UL3	Link user data 1 to 3
CONWGT	Convoy weight for current product on link
WBYVEH	Weight by vehicle for current product and for link mode
VEHWGT	Vehicle weight for current product and for link mode
UP1,UP2,UP3	Current product user data 1 to 3
VOLTOT	Total product volume on link
VEHPROD	Vehicles equivalent of current product volume on link
VEHTOT	Vehicles equivalent of total product volume on link
TRAIN	Convoys equivalent of total product volume on link
STRAIN	Convoys equivalent of total product volumes on parallel links that share the same infra structure with the mode of the current link
UM1,UM2,UM3	Link mode user data 1 to 3
UMP1,UMP2,UMP3	Link mode - current product user data 1 to 3

FUNCTIONS ON TRANSFERS

Function_Types:	**Codes:**
Operating cost	FCTnn
Delay	FDTnn
Energy consumption	FFTnn

Variables:	
VOLTOT	Total product volume on transfer
UP1-UP3	Current product user data 1 to 3
UN1-UN3	Transfer node user data 1 to 3

```
a fcl01 = ( 5690 * ump1 +  9400 * ump2 + 16920 * ump3) / wbyveh * length
a fcl02 = ( 5760 * ump1 +  9600 * ump2 + 17150 * ump3) / wbyveh * length
a fcl03 = ( 5780 * ump1 +  9940 * ump2 + 19270 * ump3) / wbyveh * length
a fcl04 = ( 6310 * ump1 + 10390 * ump2 + 17970 * ump3) / wbyveh * length
a fcl05 = ( 6390 * ump1 + 10610 * ump2 + 18230 * ump3) / wbyveh * length
a fcl06 = ( 6430 * ump1 + 11020 * ump2 + 20380 * ump3) / wbyveh * length
a fcl07 = ( 7130 * ump1 + 11700 * ump2 + 19380 * ump3) / wbyveh * length
a fcl08 = ( 7250 * ump1 + 11980 * ump2 + 19530 * ump3) / wbyveh * length
a fcl09 = ( 7300 * ump1 + 11910 * ump2 + 21860 * ump3) / wbyveh * length
a fcl10 = ( 7220 * ump1 + 11700 * ump2 + 19380 * ump3) / wbyveh * length
a fcl11 = ( 7340 * ump1 + 12450 * ump2 + 19680 * ump3) / wbyveh * length
a fcl12 = ( 7400 * ump1 + 13090 * ump2 + 21860 * ump3) / wbyveh * length
a fcl13 = ( 8370 * ump1 + 13090 * ump2 + 20320 * ump3) / wbyveh * length
a fcl14 = ( 8560 * ump1 + 13430 * ump2 + 20680 * ump3) / wbyveh * length
a fcl15 = ( 8650 * ump1 + 14000 * ump2 + 22910 * ump3) / wbyveh * length
a fcl16 = (13300 * ump1 + 16310 * ump2 + 22200 * ump3) / wbyveh * length
a fcl17 = (14140 * ump1 + 16860 * ump2 + 22705 * ump3) / wbyveh * length
a fcl18 = (14570 * ump1 + 17750 * ump2 + 25104 * ump3) / wbyveh * length
a fcl20 = length * ((1+vehwgt/wbyveh) * (0.02768 + 4.64554/(conwgt*beta/1000)
          + 10.84368/(beta/1000)) + (1/(wbyveh/1000)) * 0.39760)
a fcl21 = length * ((1+vehwgt/wbyveh) * (0.04320 +11.20169/(conwgt*beta/1000)
          + 18.54627/(beta/1000)) + (1/(wbyveh/1000)) * 1.91142)
a fcl22 = length * ((1+vehwgt/wbyveh) * (0.02765 + 4.89374/(conwgt*beta/1000)
          + 20.52866/(beta/1000)) + (1/(wbyveh/1000)) * 1.58149)
a fcl23 = length * ((1+vehwgt/wbyveh) * (0.01529 +15.27347/(conwgt*beta/1000)
          + 11.98602/(beta/1000)) + (1/(wbyveh/1000)) * 0.62926)
a fcl24 = length * ((1+vehwgt/wbyveh) * (0.05655 +21.16890/(conwgt*beta/1000)
          + 23.01850/(beta/1000)) + (1/(wbyveh/1000)) * 1.42000)
a fcl25 = length * ((1+vehwgt/wbyveh) * (0.04268 +17.41310/(conwgt*beta/1000)
          + 17.35440/(beta/1000)) + (1/(wbyveh/1000)) * 2.25310)
a fcl26 = length * ((1+vehwgt/wbyveh) * (0.01529 + 4.64554/(conwgt*beta/1000)
          + 10.84368/(beta/1000)) + (1/(wbyveh/1000)) * 0.39760)
a fcl27 = length * ((1+vehwgt/wbyveh) * (0.0 529 + 4.64554/(conwgt*beta/1000)
          + 10.84368/(beta/1000)) + (1/(wbyveh/1000)) * 0.39760)
a fcl28 = length * ((1+vehwgt/wbyveh) * (0.04486 +24.54590/(conwgt*beta/1000)
          + 52.62030/(beta/1000)) + (1/(wbyveh/1000)) * 1.11230)
a fcl30 = 267.00 * length
a fcl31 = 267.00 * length
a fcl32 = 267.00 * length
a fcl33 = 267.00 * length
a fcl34 = 267.00 * length
a fcl40 = 61.99 * length / up1
a fcl41 = 57.98 * length / up1
a fcl42 = 68.94 * length / up1
a fcl43 = 81.02 * length / up1
```

```
a fc150 = 22010.
a fc151 = 252600.
a fc152 = 89600.
a fc153 = 19320.
a fc154 = 192610.

a fd101 = (.01524 * ump1 + .01595 * ump2 + .01499 * ump3) * length
a fd102 = (.01572 * ump1 + .01828 * ump2 + .01710 * ump3) * length
a fd103 = (.01634 * ump1 + .01832 * ump2 + .01862 * ump3) * length
a fd104 = (.01585 * ump1 + .01631 * ump2 + .01616 * ump3) * length
a fd105 = (.01631 * ump1 + .01789 * ump2 + .01825 * ump3) * length
a fd106 = (.01695 * ump1 + .01938 * ump2 + .01980 * ump3) * length
a fd107 = (.01712 * ump1 + .01783 * ump2 + .01927 * ump3) * length
a fd108 = (.01761 * ump1 + .01938 * ump2 + .02128 * ump3) * length
a fd109 = (.01761 * ump1 + .02079 * ump2 + .02228 * ump3) * length
a fd110 = (.01805 * ump1 + .01912 * ump2 + .02315 * ump3) * length
a fd111 = (.01852 * ump1 + .02058 * ump2 + .02488 * ump3) * length
a fd112 = (.01916 * ump1 + .02198 * ump2 + .02625 * ump3) * length
a fd113 = (.01988 * ump1 + .02128 * ump2 + .02591 * ump3) * length
a fd114 = (.02041 * ump1 + .02273 * ump2 + .02755 * ump3) * length
a fd115 = (.02101 * ump1 + .02410 * ump2 + .02882 * ump3) * length
a fd116 = (.02513 * ump1 + .02725 * ump2 + .03425 * ump3) * length
a fd117 = (.02571 * ump1 + .02865 * ump2 + .03546 * ump3) * length
a fd118 = (.02625 * ump1 + .02976 * ump2 + .03650 * ump3) * length
a fd120 = (length/ul2) * ( 1 + (1.1/cap) * (strain*19.23) + 4.144 * (((strain
          *19.23)/cap)^6))
a fd130 = 0.076923 * length
a fd131 = 0.125 * length
a fd140 = .08295 * length
a fd141 = .06897 * length
a fd142 = .08100 * length
a fd143 = .07791 * length
a fd150 = ( 0.050000 ) * ( voltot.le.cap ) + ( 0.050000 + 2 * ( voltot-cap ))
          * ( voltot.gt.cap)
a fd151 = ( 10.00000 ) * ( voltot.le.cap ) + ( 10.00000 + 2 * ( voltot-cap ))
          * ( voltot.gt.cap)
a fd152 = ( 10.00000 ) * ( voltot.le.cap ) + ( 10.00000 + 2 * ( voltot-cap ))
          * ( voltot.gt.cap)

a fd160 = ( 0.052083 ) * ( voltot.le.cap ) + ( 0.052083 + 2 * ( voltot-cap ))
          * ( voltot.gt.cap)
a fd161 = ( 0.021142 ) * ( voltot.le.cap ) + ( 0.021142 + 2 * ( voltot-cap ))
          * ( voltot.gt.cap)
a fd162 = ( 10.00000 ) * ( voltot.le.cap ) + ( 10.00000 + 2 * ( voltot-cap ))
          * ( voltot.gt.cap)
```

```
a fdl70 = ( 0.050505 ) * ( voltot.le.cap ) + ( 0.050505 + 2 * ( voltot-cap ))
          * ( voltot.gt.cap)
a fdl71 = ( 10.00000 ) * ( voltot.le.cap ) + ( 10.00000 + 2 * ( voltot-cap ))
          * ( voltot.gt.cap)
a fdl72 = ( 0.004021 ) * ( voltot.le.cap ) + ( 0.004021 + 2 * ( voltot-cap ))
          * ( voltot.gt.cap)
a fdl80 = ( 0.032680 ) * ( voltot.le.cap ) + ( 0.032680 + 2 * ( voltot-cap ))
          * ( voltot.gt.cap)
a fdl81 = ( 0.017241 ) * ( voltot.le.cap ) + ( 0.017241 + 2 * ( voltot-cap ))
          * ( voltot.gt.cap)
a fdl82 = ( 0.007052 ) * ( voltot.le.cap ) + ( 0.007052 + 2 * ( voltot-cap ))
          * ( voltot.gt.cap)

a fct01 = 54060.
a fct02 = 54060.
a fct03 = 40480.
a fct04 =  5520.
a fct05 = 52010.
a fct10 = 82722.
a fct50 = 0.0
a fct60 = 0.0
a fct70 = 500000.

a fdt01 = .01815
a fdt02 = .09091
a fdt03 = .09091
a fdt04 = .00945
a fdt05 = .01815
a fdt10 =  (0.090909) * (voltot.le.100) + (0.090909 + 2 * (voltot - 100))
          * (voltot.gt.100)
a fdt11 =  (10.00000) * (voltot.le.10) + (10.00000 + 2 * (voltot - 10))
          * (voltot.gt.10)
a fdt12 =  (10.00000) * (voltot.le.10) + (10.00000 + 2 * (voltot - 10))
          * (voltot.gt.10)
a fdt20 =  (0.029070) * (voltot.le.500) + (0.029070 + 2 * (voltot - 500))
          * (voltot.gt.500)
a fdt21 =  (0.065789) * (voltot.le.500) + (0.065789 + 2 * (voltot - 500))
          * (voltot.gt.500)
a fdt22 =  (10.00000) * (voltot.le.10) + (10.00000 + 2 * (voltot - 10))
          * (voltot.gt.10)
a fdt30 =  (0.333333) * (voltot.le.500) + (0.333333 + 2 * (voltot - 500))
          * (voltot.gt.500)
a fdt31 =  (10.00000) * (voltot.le.10) + (10.00000 + 2 * (voltot - 10))
          * (voltot.gt.10)
a fdt32 =  (0.034722) * (voltot.le.500) + (0.034722 + 2 * (voltot - 500))
```

```
          * (voltot.gt.500)
a fdt40 =  (0.006916) * (voltot.le.1500) + (0.006916 + 2 * (voltot - 1500))
          * (voltot.gt.1500)
a fdt41 =  (0.018149) * (voltot.le.1500) + (0.018149 + 2 * (voltot - 1500))
          * (voltot.gt.1500)
a fdt42 =  (0.009452) * (voltot.le.1500) + (0.009452 + 2 * (voltot - 1500))
          * (voltot.gt.1500)
a fdt50 = 0.0
a fdt60 = 0.0
a fdt70 = 10000.
```

Appendix 2

In what follows, the definition of the variables is the same as in Appendix 1.

```
a fcl1   =(((1 + vehwgt / wbyveh) * .0431 / (beta * conwgt)) + .0026 /
           wbyveh) * length
a fcl2   =(((1 + vehwgt / wbyveh) * .09860002 / (beta * conwgt)) +
           .002085 / wbyveh) * length
a fcl3   =0
a fcl4   =.0195 * length / wbyveh
a fcl5   =.0245 * length / wbyveh
a fcl6   =.00975 * length / wbyveh

a fdl1   =0
a fdl2   =.992 * ((1 + vehwgt / wbyveh) / (conwgt * beta)) * .0182 *
          length * (1 + .055 * strain + 6.17999982 * ((strain / cap)
          ^ 6))
a fdl3   =.992 * ((1 + vehwgt / wbyveh) / (conwgt * beta)) * .0333 *
          length * (1 + .055 * strain + 6.17999982 * ((strain / cap)
          ^ 6))
a fdl4   =.00166 * length / wbyveh
a fdl5   =.002 * length / wbyveh
a fdl6   =.25 * length / wbyveh
a fdl10  =5 + 5 * ((voltot / cap) ^ 3)

a fct1   =5
a fct2   =.01 * (1 + un1)

a fdt1   =5
a fdt2   =1 * (1 + up3)
```